A Tree Clustering Algorithm For Web Graphs

D. Auber and M. Delest

LaBRI-Université Bordeaux 1, 351 Cours de la Libération, 33405 Talence, France
{maylis,auber}@labri.u-bordeaux.fr

Abstract. We present a new tree clustering algorithm based on combinatorial statistics on trees. Using well-known measures on trees giving the number of leaves of a subtree or the number of siblings of a node, we design a parameter that can be used to detect irregularities in a tree. We obtain a clustering approach for trees by implementing classical sta-tis-tical tests on this parameter, thus providing well balanced drawings of trees offering better aspect ratios which can be useful when dealing with large, irregular hierarchical data. Our algorithm is linear in time and can thus be applied to large data structures. Moreover larger is the structure better is the precision of the statistical tools.

1 Introduction

With the increasing computing power and storage capacity of computers, Information Visualization researchers can tackle problems dealing with larger amount of data. Thousands of servers are accessible from the everyday computer through the Internet. A crucial task in this context is to bring information closer to the user through well-designed interfaces. Typically, the extracted information can be presented as a graph with a few thousands nodes. In many situations, such as the web, the exploration can be based on a tree extracted from the actual graph. Many visualization systems dealing with large graph structures have already been proposed. NicheWorks [13] and daVinci [5] are examples. Those systems focus on scalability and ergonomy issues related to the user interface and implement specific techniques of navigation. Other approaches avoid presenting the overall structure and rely on a dynamic view of the graph to support web exploration (see [8], for example). Another approach is to offer automatic clustering of the graph to maintain the visible structure within some bounds. In general, a cluster will gather similar elements. However, the notion of *similarity* needs to be adapted according to the task the user is performing on the data. Information Visualization applications often make use of tree representations. This situation sometimes comes from the data itself, or is created by systematically extracting spanning trees from the (graph) data. In this paper, we assume that the structure to visualize is a tree. Our main result is a hierarchical clustering algorithm for large trees based on node metrics. Our approach refines that presented in [6, 7] and gives better results. The clustering algorithm can be interacted with and can be seen as a folding feature implemented in a visualization system that seeks for irregular sub-structures, that is, those sub-structures having a metric value out of a given confidence interval.

Figure 1 shows the tree representation of a data structure generated by a parallel machine compiler. In this case, the tree is directly obtained from the compiler. Observe that nodes have at most four children nodes. This is a direct consequence of the fact that the machine uses four processors. This example thus justifies that we study the visualization of trees where nodes have a bounded outgoing degree (number of children).

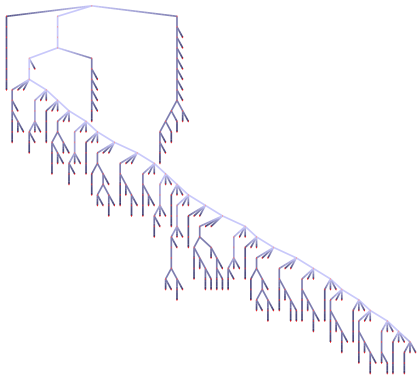

Fig. 1. A tree from the data structure of a parallel compiler.

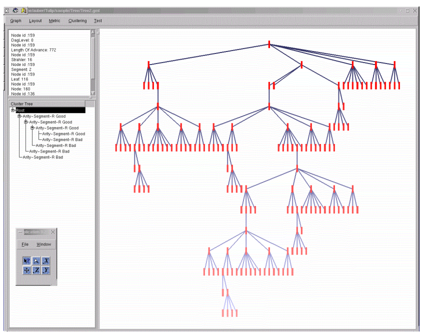

Fig. 2. Tulip : a view of the software

Consider now the tree map of a web site. The tree could have been obtained from a web grabber. In this case, the tree will most certainly not contain long paths of nodes having a single child node. Indeed, this would correspond to web pages containing only one outlink to a page having only one outlink and so on. A *segment* in a tree is such a sequence of nodes having outdegree exactly equal to one (except for the last one which can have outdegree zero – if it is a leaf node). The average maximal segment length, at least for examples such as the web, seems to be rather low (except maybe for links to slide presentations). Kleinberg quoted this fact in [10]. Figure 13 gives an example of a web graph.

Those observations are of importance if one considers implementing automatic procedures based on the statistical behaviour of metrics on trees. More precisely, when applying the folding procedure to large trees, one cannot simply use knowledge on the statistical distribution of the class of *all* trees, but must first try to get at the distribution of the class of trees with segments of bounded length. In this paper, we describe the statistical distribution of the class of trees where nodes have a fixed maximum outdegree and/or a maximum segment length. We then show how this knowledge can be used to implement a folding procedure on large trees in order to hide irregular components.

Let us describe an example where this knowledge can be useful. When browsing a web site, a user might be interested to locate a page listing all available on-line services. Let us agree to call such a node a directory node (in a web site). In the tree structure describing the web site, this node will typically appear as a node with an unusually high number of children nodes. Assuming that the user has access to the layout of the tree, it might be easy for him/her to locate a directory node if all its children nodes are leaves. This will be the case if a node gives access to phone numbers of employees in a company. However, if the children nodes are not all leaf nodes but rather contain links to other pages, the visual inspection of the site might not reveal the fact that a node actually is a directory node. In this case, the user might be helped by a tool scanning the structure and looking for nodes with irregularities – with an unusually high number of children nodes, in our example.

The paper first looks at the distribution of the number of leaves in a tree where nodes have a fixed maximum outdegree and/or segment length. Using standard combinatorial techniques, we prove that this distribution is Gaussian and differs from that for the class of all trees (Sect. 3). Using this result, we give an algorithm that hierarchically cluster the tree and, as a consequence, emphasizes *regularity* or *irregularity* sub-structures (Sect. 5). Because our algorithm has linear complexity, it can be implemented in an interactive environment (Sect. 4). An implementation of the algorithm has been written in C++ and integrated in a graph visualization software called Tulip [1] developed at LaBRI.

2 Definitions and notations

We begin by giving usual definitions on trees. We then state basic facts needed in the forthcoming sections (section 3). Let T be a planar [1]tree with $|T|$ nodes and $|T|_L$ leaves. Let s be a node of T. We will denote by T_s the subtree of T rooted at s. The arity of s is its number of children (or successors) in T denoted by $deg_T^+(s)$. Thus, if s is not the root than its degree in T is $deg_T^+(s)+1$.

Definition 1. *A segment of length n in T is a path $C = (s_1, s_2, ..., s_n, s_{n+1})$ such that*

$$\forall\ i \in [1,n], deg_T^+(s_i) = 1.$$

Its length will be denoted by $|C|$. The segment C is maximal if $deg_T^+(s_{n+1}) \neq 1$ and s_1 is the root or $deg_T^+(father(s_1)) \neq 1$.

The set of maximal segments of a tree T is denoted by $\mathcal{S}(T)$. In what follows, the algorithm is based on the maximum of all the segment lengths in T that is λ_T and also the maximal node arity, denoted by α_T. The set of planar trees T such that $\lambda_T = r$ (resp. $\alpha_T = r$) will be denoted by L_r (resp. A_r). We note $B_{r,p}$ the set $A_r \cap L_p$.

As usual, the clustering process is described using a partition tree. Let us recall the definition of a partition. Let $\mathcal{P}(E)$ be a partition of a set E of n

[1] That is, the successors of a node are assumed to be totally ordered.

elements. It is a set of k parts $\{w_i\}_{i=1..k}$ of E such that $\bigcup_{i=1..k} w_i = E$ and $\forall i, j \in [1, k], i \neq j w_i \cap w_j = \varnothing$. Then, the set $W = \{w_i\}_{i=1..k}$ can be viewed as a set of elements. Suppose that the partition process can be recursively applied to W, then we get a new partition $\mathcal{P}(W)$.

Definition 2. *A partition tree is a tree in which the root represents the initial set and each node represents an element (a part) in the recursive partition process. The inclusion relation is given by the edges.*

Note that, to each level in the partition tree corresponds a partition of the original set.

Before proving the results of section 3, we illustrate elementary concepts from enumerative combinatorics on a simple example. More precisely, we shall make use of the so-called object grammars [3], providing a generic and powerful approach to generating functions and asymptotic estimations [11]. One important feature of object grammars is to enable the use of "visual equations" to express functional identities on generating functions. Other techniques could also be used but object grammars are more *visual.* Let $\mathcal{T}$ be the set of planar trees. Clearly the figure 3 shows the natural decomposition of a planar tree. A tree is just a node or it is made with an edge with two subtrees. Then, object grammars method allows writing the recursive equation :

$$\mathcal{T} = \bullet + \phi(\mathcal{T}, \mathcal{T}).$$

In this equation, ϕ represents an object operation. Let O_1 and O_2 be planar trees, then ϕ consists in gluing O_1 with O_2 at each end of an edge. If one wants the generating function for planar trees, this fact can be translated immediately into an equation that is here $t(x) = 1 + x * t(x)^2$. Solving it, gives the well-known number of trees having n nodes that is $(n-1)^{th}$ Catalan number.

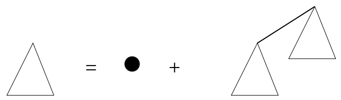

Fig. 3. An object grammar for planar trees.

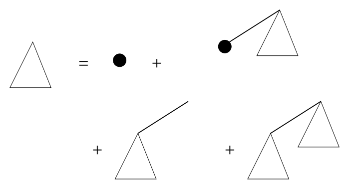

Fig. 4. An object grammar counting leaves for planar trees.

Suppose now that one wants to refine this enumeration. Let us define a new parameter π that is here the number of leaves in a planar tree. The figure 4 shows a new object grammar in which all the leaves appear. We have a recursive equation

$$\mathcal{T} = \bullet + \phi(\bullet, \mathcal{T}) + \phi_1(\mathcal{T}) + \phi(\mathcal{T}, \mathcal{T}).$$

We can deduce the equation $g(x,z) = xz + xzg(x,z) + x * g(x,z) + x * g(x,z)^2$ in which $g(x,z) = \sum_{n\geq 0, m\geq 0} g_{n,m} x^n z^m$ and $g_{n,m}$ is the number of planar trees having size n and m leaves. It is well known that

$$g_{n,m} = \frac{1}{n-1}\binom{n-1}{m}\binom{n-1}{m-1}.$$

Thus, using such a method, we can compute distribution on parameters on trees every time that the problem remains in a certain class (larger than algebraic).

If a parameter is algebraic then a constructive theorem from Dmróta [11] gives the exact law for the random variable associated to the parameter that means here the probability of having a tree T of size n having m leaves. This theorem holds as soon as n is large enough that is in our cases more than 10. The method is the following :

- compute the main singularity of $g(x,1) = \phi(g,x,1)$,
- compute the mean μ and standard deviation σ using Dmróta formula.

Then the random variable π is a Gauss variable with mean μ and standard deviation σ. In our example, the main singularity of $g(x,1)$ is $\chi = 1/4, \gamma = 1/2$. Then, from the Dmróta formula, the random variable *number of leaves* in the tree of size n has a Gauss distribution with mean $\mu = n/2$ and standard deviation $\sigma = \sqrt{n/8}$.

From this point, our method is straightforward. We use classical statistical test [9]. We construct the confidence interval for the random variable π of mean μ and standard deviation σ that is for a given level u_ϵ from Gauss table : $I(\pi) = [\mu - u_\epsilon\sigma, \mu + u_\epsilon\sigma]$. In our example, for a given n, suppose that we choose $u_\epsilon = 1.96$ then $\epsilon = 0.05$. This means that 5 percents of the trees of size n have a number of leaves that falls outside the interval

$$I_n(\pi) = [n/2 - u_\epsilon\sqrt{n/8}, n/2 + u_\epsilon\sqrt{n/8}].$$

Then, for a given subtree of size n, if the π value falls outside $I_n(\pi)$ for a given ϵ then the subtree is folded : it is too far from the average tree. The fold tool in the Latour software [7] is an implementation of this result. We prove in the following sections that the number of segments and the arity of the input tree drastically modify the interval. Moreover, we show that, using this refinement, clusters can be constructed for the input tree.

3 Combinatorial results

In this section, we apply the combinatorial tools describe in the previous section in order to construct confidence interval for clustering the $B_{r,p}$ trees according to the leaves number. Similar results can be obtained for L_k and A_r or others algebraic parameters. An object grammar for $B_{r,p}$ is displayed on figure 5. The objects are the following :

- a triangle represents a tree of $B_{r,p}$,

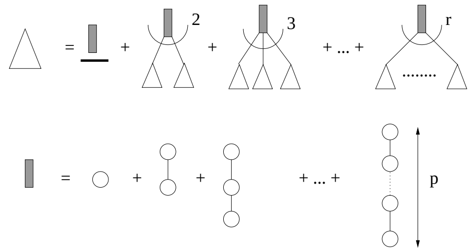

Fig. 5. An object grammar for $B_{r,p}$ planar trees.

- a black rectangle represents segments of length p,
- a white circle represents any vertex,
- an underlined object contains one leaf more than the original object.

The equations mean that a $B_{r,p}$ tree is obtained by constructing a tree without segment and of maximum arity r and then substituting to each vertex a segment of length at most p. The underline mark means that each leaf is at the bottom of a final segment. Let us define the generating function $B_{r,p}(x,z) = \sum_{n\geq 0, k\geq 0} b_{n,k} x^n z^k$ where $b_{n,k}$ is the number of $B_{r,p}$ tree having n nodes and k leaves. From figure 5, we get a system of equations for which $B_{r,p}(x,z)$ is a solution

$$\begin{cases} B_{r,p}(x,z) = S_p(x) z + x \sum_{i=2}^{r} B_{r,p}^i \\ S_p(x) = \sum_{i=1}^{p+1} x^i \end{cases}.$$

Theorem 1. *The generating function $B_{r,p}(x,z)$ verifies the equation $B_{r,p}(x,z) = F(B_{r,p}, x, z)$ with*

$$F(B_{r,p}, x, z) = \frac{x\left(1-x^{p+1}\right)\left(1-{B_{r,p}}^{k+1}\right)}{(1-x)\left(1-B_{r,p}\right)} - \frac{x\left(1-x^{p+1}\right)\left(1-z+B_{r,p}\right)}{1-x}$$

The main singularity of $B_{r,p}(x,1)$ can be obtained [4] by solving

$$\begin{cases} B = F(B,x,1) \\ \frac{\partial}{\partial B} F(B,x,z) = 1 \end{cases}$$

that is , we need to

- solve the equation $1 - 2B + B^{r+2} + B^{r+1}k - B^{r+2}r = 0$
- get the minimal non zero root β_r
- for each r and each p, substitute the value β_r in

$$x\left(-1+x^{p+1}\right)\left(1-{\beta_r}^{r+1}-\beta_r+{\beta_r}^2\right)+\beta_r\left(-1+\beta_r\right)(1-x)$$

- for each r and each p, get the minimal non zero root $\xi_{r,p}$

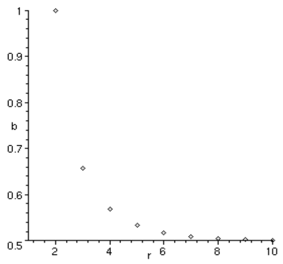

Fig. 6. Plot for β_r.

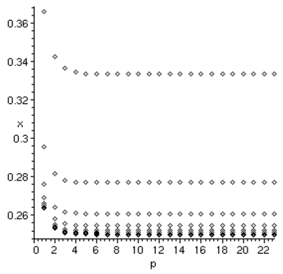

Fig. 7. Plot for $\xi_{r,p}$.

In this case, exact values cannot be found but $(\xi_{r,p}, \beta_r)_{r\geq 2, p\geq 1}$ can be computed. Of course, it converges to the value (χ, γ) of the section 2 (see figures 6 and 7). Then, applying results in[11] to $B_{r,p}$, for all values of r and p, we compute the mean and standard deviation of the number of leaves in a $B_{r,p}$ tree of size n. The mean value is obtained by computing

$$\mu_{B_{r,p}}(n) = \frac{\frac{\partial}{\partial z} F(B,x,z)}{x \frac{\partial}{\partial x} F(B,x,z)} n$$

evaluated at $x = \xi_{r,p}$,$B = \beta_r$, $z = 1$. The identity giving the standard deviation is more complex and is omitted. In the figure 8, the Gauss distributions are

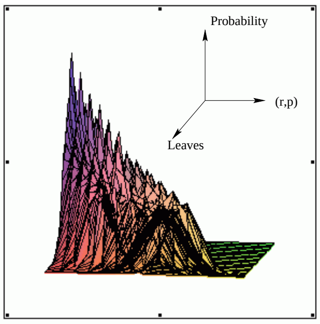

Fig. 8. Plot for the leaves distribution for all r, p and $n = 100$.

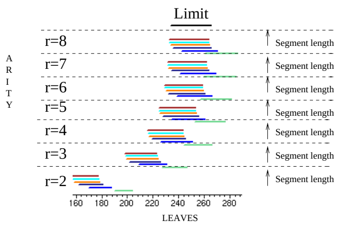

Fig. 9. The confidence interval for the leaves number with *epsilon* $= 0.05$ and $n = 500$.

plotted for all values of r, p and $n = 100$. One can see that the curves greatly differ from one another. Every peak corresponds to the mean of the *number of leaves* for a given r and a given p. It means that, for the parameter leaves, arity and segment length are significant even for trees having 100 nodes. In the

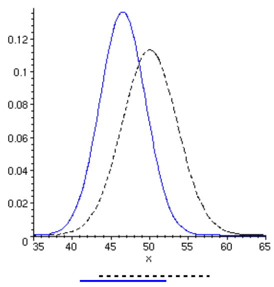

Fig. 10. The Gauss distribution for the number of leaves of $B_{4,3}$ having 100 nodes.

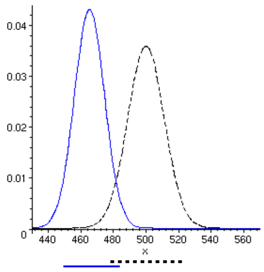

Fig. 11. The Gauss distribution for the number of leaves of $B_{4,3}$ having 1000 nodes.

figure 10 and 11, we plot two curves one corresponding to $\mathcal{T}$ (dashed line) and the other to $B_{4,3}$ (solid line). The confidence intervals for $\epsilon = 0.05$ are drawn at the bottom of each curve. One can see that for $n = 100$, the distributions are slightly different but this difference increases when $n = 1000$.

So, suppose that a subtree of size 1000 has 460 leaves. The leaves number is not in the confidence interval if the subtree comes from $\mathcal{T}$ but it is if the input tree is known to have maximum arity 4 and maximal length segment 3. Thus, in our algorithm, for such a value, a planar subtree will be clustered while a subtree of a $B_{4,3}$ input tree will not be clustered. In fact larger is n, greater is the difference and better is the refinement.

A computation on the relative error shows that with an error of 0.001, if $r \geq 9$ and $p \geq 7$ then $\mu_{B_{r,p}} = n/2$ and $\sigma_{B_{r,p}} = \sqrt{n/8}$. Thus, in these cases, the tree can be considered as a tree in $\mathcal{T}$. For $\epsilon = 0.05$ and $n = 500$, the confidence intervals are plotted on figure 9. The upper segment figures out the limit one. Note that $\mu_{B_{r,p}} = \theta_{r,p} n$ and $\sigma_{B_{r,p}} = \delta_{r,p}\sqrt(n)$ where $\theta_{r,p}$ and $\delta_{r,p}$ are constant. Thus in the algorithm of section 4, we just need to keep the tables $\{\theta_{r,p}\}_{2 \leq r \leq 8, 1 \leq p \leq 6}$ and $\{\delta_{r,p}\}_{2 \leq r \leq 8, 1 \leq p \leq 6}$.

4 Algorithms

In this section, we present an algorithm that takes as input a tree and outputs a partition tree (see section 2) and views of the clustering on the initial tree. The partition tree can then be used as a navigation tool offering partial views of the original tree. The user can thus go through a step-by-step analysis of the data. We create the partition tree using previous theoretical results based either on the arity of the nodes, or the segment lengths, or a combination of the both (section 3). In all the three cases, the partition tree can be computed in linear time (according to the size of the original tree). This provides an efficient technique of navigation that can be applied to large data structures such as the web trees or trees obtained from parallel compiler. In the following, we call metrics any parameter computed on the input tree. The partition tree has the following properties :

- it is a binary tree,
- its root is associated to the set of all the nodes of the input tree,
- two children of any internal node v represent two parts of a bipartition of the nodes set represented by v .

Let v be a node of the partition tree, we will denote by $\mathcal{V}(v)$ the set of nodes represented by v. Let T be a tree of $\mathcal{T}$. Our process consists in applying successively a tree-pruning algorithm that is removing a subtree except its root. We do a depth first search on the subtrees that are induced by the prefix order on the nodes of T. At each step, we separate the nodes of the input tree in two parts. Let v be the current node of the partition tree, $\mathcal{V}(s)$ induce a cluster in the initial tree T say $\mathcal{C}(T)$. Thus, we identify in $\mathcal{C}(T)$ the subtrees set $P(T)$ that we have to prune using the following properties:

- the subtree does not contain a node s such that T_s has already been cut during this pruning step,
- the leaves number of the subtree root is not in the confidence interval,

Then, in the partition tree, we add two children v_1 and v_2 to v : $\mathcal{V}(v_1)$ is the set of nodes that are in the subtrees of $P(T)$ and $\mathcal{V}(v_2) = \mathcal{V}(v) \backslash \mathcal{V}(v_1)$. Next step, the tree to be clustered will be the tree obtained by pruning all the subtrees of $P(T)$ in $\mathcal{C}(T)$ and the current tree partition node will be v_2.

Note that when only one metric (as in this paper) is used in the pruning process, the vertex s_1 will not vary in the next steps. The algorithm ends when the pruned tree is equal to the entry. The figure 12 shows a first version of this algorithm.

The maximum number of parts on the initial nodes set is equal to the nodes number of the tree$|T|$. So, the maximum number of iteration is $|T|$ and the complexity of the previous algorithm is $O(|T|^2)$. However, this version is a simplified version of the final algorithm. In fact, if the metrics can be computed through a synthesized attribute [2] and their values are coded with a constant amount of words, then, using the following tricks, the complexity of the algorithm become linear according to $|T|$. First, the metrics computation process must be included in the algorithm and their values must be kept. This can be done by only computing the value on each node touched on the current step, and, during the pruning operation, computing the subtree root metrics. Then, the goal is to build the full partition tree with only one depth first search. For that, using an integer, we encode the inclusion during the ascending process in the depth first search. Let T_s be a subtree of root s. As example, in this paper, we only used four metrics $|T_s|$, $|T_s|_L$ λ_{T_s} and α_{T_s}. Their characteristics agree with the previous remarks.

Three algorithms have been implemented. All are based on the number of leaves and nodes of the subtree. Statistical tables of means and standard deviations are stored indexed by arity and segment length. In this way, deciding to

```
Partition Tree Clustering(Tree T){
  Tp a partition tree with R as root
  R ← {All nodes of T}
  T' a tree
  P a set of nodes
  T' ← T
  REPEAT{
      T ← T'
      P ← ∅
      compute measures (m_1, m_2..m_k) on T'
      Remove flag from all nodes of T'
      E a sequence of nodes build with a depth first search on T'
      FOR all elements x of E DO
        IF x doesn't have descents with a flag THEN
          IF the measures(m_1(x), m_2(x)..m_k(x)) are not in the confidence intervals THEN
            P ← P ∪ (A_x \ {x})
            T' ← T' \ (T_x \ {x})
          END IF
        END IF
      END FOR
      S1,S2 two new nodes of Tp such that R is their father
      S1 ← P
      S2 ← {All nodes of T'}
      R ← S2
    } UNTIL T==T'
  RETURN Tp}
```

Fig. 12. A first version of the algorithm.

prune or not will remains in O(1). The first algorithm based on the arity will prune the tree according to the confidence interval computed from the statistical table indexed by its maximum arity. We gradually remove from our starting tree all the abnormally ramified parts (too or too much little). The others are similar using statistical tables indexed by length segment and arity/segment length. In this algorithm , many metrics on tree can be used that we have not yet explored.

5 Experimental results

The algorithm was implemented in the Tulip software and the metrics too. This software has navigation tools for the partition tree, enlightens cut subtrees at each step, menus for activating the metrics, popup window for choosing the value of ϵ in the confidence interval, see figure 2. Several graph-drawing algorithms are available in Tulip. In the figure of this paper, we use the Reingold-Tilford algorithm [12].

One of the main features is that it enlightens step-by-step the partition of the tree since in Latour [7] there is only two position fold or unfold. Also, the final result is different because the confidence intervals are more precise.

Below, we only show the result on a part of the web tree of our laboratory (see figure 13, 1165 nodes). We use the two metrics arity and segment length. First note that at each pruning step, independent directories are removed. It induced nicer and nicer drawing because a large width of a tree in regard of its height gives bad angular resolution in classical tree drawing and in such situation directories are enlarging the width. The number of step of the algorithm is seven. It leads up to views that are understandable and with slow changes between them. The figures 13 to 18 give the successive cluster. In step one, the pruning

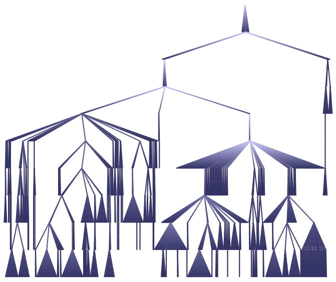

Fig. 13. Starting tree

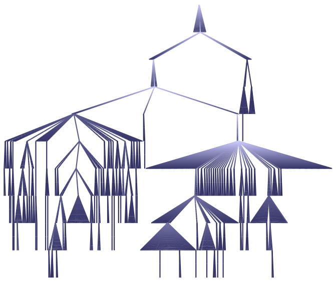

Fig. 14. Step one

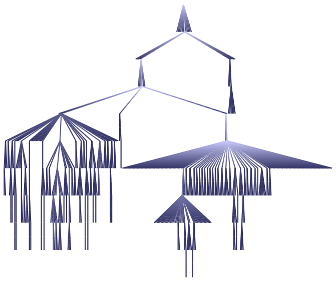

Fig. 15. Step two

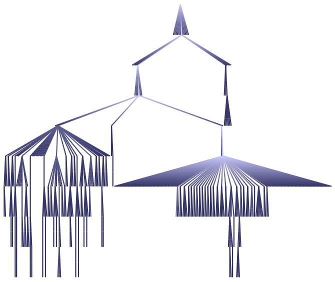

Fig. 16. step three

process enlightens the big directory on the right side by cutting small directories at the bottom and too long segments (see on the right upper side). In the step three, the tree presents clearly two subtrees looking as directory. Note that the right subtree is the directory of all the persons in our laboratory and the left one is the entry of one of the main administration directory that is information for

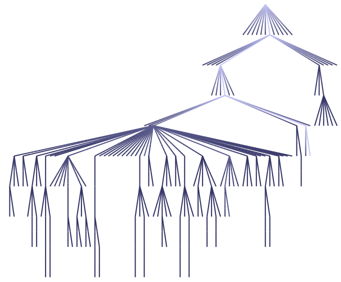

Fig. 17. Step four

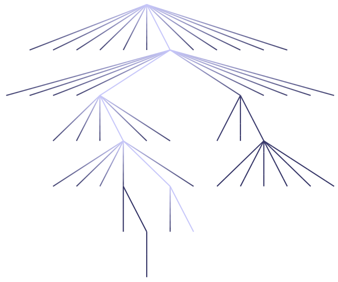

Fig. 18. Step five

students. Automatically, the algorithm cut the more compact that is the right side one.

At the final, the interest of such statistical tools is in the stability of the results and in the fact that finding *irregularity* in a big tree is of linear complexity. Moreover, many algebraic parameters are well-known on trees. We plan to explore their usage for clustering.

References

1. Auber D. Tulip : A software for graph drawing experimentations. Technical Report XX, LaBRI, 2000.
2. Knuth D.E. Semantics of context-free languages. *Maths. Sys.*, 2:127–145, 1968.
3. Dutour I. and Fédou J.M. Object grammars and random generation. *Discrete Math. Theor. Comput. Sci.*, 2:47–61, 1998.
4. Flajolet P. and Sedgewick R. The average case analysis of algorithms: Counting and generating functions. Technical Report 1888, INRIA, 1993.
5. Fröhlich M. and Werner M. Demonstration of the interactive graph visualization davinci. In *DIMACS Workshop on Graph Drawing'94*, volume 894 of *Lectures Notes in Computer Science*. Springer-Verlag, 1995.
6. Herman, I., Delest, M., and Melanon, G. Tree visualisation and navigation clues for information visualisation. *Computer Graphics Forum*, 17(2):153–165, 1998.
7. Herman, I., Melanon, G., Ruiter, B.d., and Delest, M. Latour - a tree visualization system. In J. Kratochvil, editor, *Symposium on Graph Drawing GD'99*, Lecture Notes in Computer Science. Springer-Verlag, 1999.
8. Huang, M.L., Eades, P., and Cohen, R.F. Webofdav - navigating and visualizing the web on-line with animated context swapping. In *7th World Wide Web Conference*, pages 636–638. Elsevier Science, 1998.
9. Kendall M.G. and Stuart A. *The advanced theory of statistics.* Charles Griffin & company ltd, 1966.
10. Kleinberg J., Kumar S.R., Raghavan P., Rajagopalan S., and Tomkins A. The web as a graph: Measurements, models and methods. In *International Conference on Combinatorics and Computing*, 1999.
11. Dmróta M. Asymptotic distributions and a multivariate darboux method in enumeration problems. *J. of Combinatorial theory A*, 67:169–184, 1994.

12. Reingold E.M. and J.S. Tilford J.S. Tidier drawing of trees. *IEEE Transactions on Software Engineering.*, 2, 1981.
13. G.J. Wills. Niche works - interactive visualization of very large graphs. In *Symposium on Graph Drawing GD'97*, volume 1353 of *Lectures Notes in Computer Science*, pages 403–414. Springer-verlag, 1997.